Multibody System Simulation with SimMechanics

Michael Schlotter

May 2003

Abstract

This paper describes, how to simulate the dynamics of multibody systems with SimMechanics, a toolbox for the Matlab/ Simulink environment. The novel Physical Modeling method is compared with traditional ways to represent mechanical systems in Matlab, and some interesting mathematical aspects of the implementation of SimMechanics are examined.

1 Introduction

Simulating the dynamics of multibody systems is a common problem in engineering and science. Various programs are available for that task which are either symbolical computation programs to derive and solve the dynamical equations of motion, or numerical programs which compute the dynamics on the basis of a 3D-CAD model or by means of a more abstract representation, e.g. a block diagram.

As an add-on for the GUI-based simulation environment Simulink, SimMechanics falls into the last category. Mechanical systems are represented by connected block diagrams. Unlike normal Simulink blocks, which represent mathematical operations, or operate on signals, *Physical Modeling* blocks represent physical components, and geometric and kinematic relationships directly. This is not only more intuitive, it also saves the time and effort to derive the equations of motion.

SimMechanics models, however, can be interfaced seamlessly with ordinary Simulink block diagrams. This enables the user to design e.g. the mechanical and the control system in one common environment. Various analysis modes

and advanced visualization tools make the simulation of complex dynamical systems possible even for users with a limited background in mechanics.

2 Functionality of the Toolbox

This section provides an overview about SimMechanics. The block set is described briefly, as well as the different analysis modes and visualization options. More details about these topics can be found in [Mat02].

2.1 Physical Modeling Blocks

As already mentioned, the SimMechanics blocks do not directly model mathematical functions but have a definite physical (here: mechanical) meaning. The block set consists of block libraries for bodies, joints, sensors and actuators, constraints and drivers, and force elements. Besides simple standard blocks there are some blocks with advanced functionality available, which facilitate the modeling of complex systems enormously. An example is the Joint Stiction Actuator with event handling for locking and unlocking of the joint. Modeling such a component in traditional ways can become quite difficult. Another feature are Disassembled Joints for closed loop systems. If a machine with a closed topology is modeled with such joints, the user does not need to calculate valid initial states of the system by solving systems of nonlinear equations. The machine is assembled automatically at the beginning of the simulation.

All blocks are configurable by the user via graphical user interfaces as known from Simulink. The option to generate or change models from Matlab programs with certain commands is not implemented yet. It might be added in future releases. It is possible to extend the block library with custom blocks, if a problem is not solvable with the provided blocks. These custom blocks can contain other preconfigured blocks or standard Simulink S-functions.

Standard Simulink blocks have distinct input and output ports. The connections between those blocks are called *signal lines*, and represent inputs to and outputs from the mathematical functions. Due to Newton's third law of actio and reactio, this concept is not sensible for mechanical systems. If a body A acts on a body B with a force $\mathbf{F}$, B also acts on A with a force $-\mathbf{F}$, so that there is no definite direction of the signal flow. Special *connection lines*, anchored at both ends to a *connector port* have been introduced with this toolbox. Unlike signal lines, they cannot be branched, nor can they be connected to standard blocks. To do the latter, SimMechanics provides Sensor and Actuator blocks. They are the interface to standard Simulink models.

Actuator blocks transform input signals in motions, forces or torques. Sensor blocks do the opposite, they transform mechanical variables into signals.

2.2 Types of Analysis

SimMechanics provides four modes for analyzing mechanical systems:

Forward Dynamics calculates the motion of the mechanism resulting from the applied forces/ torques and constraints.

Inverse Dynamics finds the forces/ torques necessary to produce a specified motion for open loop systems.

Kinematics does the same for closed loop systems by including the extra internal invisible constraints arising from those structures.

Trimming searches for steady or equilibrium states of a system's motion with the Simulink `trim` command. It is mostly used to find a starting point for linearization analysis.

Generally it is necessary to build a separate model for each type of analysis, because of the different mechanical variables which need to be specified. In Forward Dynamics mode, the initial conditions for positions, velocities, and accelerations, as well as all forces acting on the system are required to find the solution. To use the Inverse Dynamics or the Kinematics mode, the model must specify completely the positions, velocities, and accelerations of the system's independent degrees of freedom.

2.3 Visualization Tools

SimMechanics offers two ways to visualize and animate machines. One is the build-in *Handle Graphics* tool, which uses the standard Handle Graphics facilities known from Matlab with some special features unique to SimMechanics. It can be used while building the model as a guide in the assembly process. If a body is added or changed in the block diagram, it is also automatically added or changed in the figure window. This gives immediate feedback and is especially helpful for new users or for complex systems. The visualization tool can also be used to animate the motion of the system during simulation. This can be much more expressive than ordinary plots of motion variables over time. The drawback is a considerably increased computation time if the animation functionality is used.

The bodies of the machine can be represented as *convex hulls*. These are surfaces of minimum area with convex curvature that pass through or surrounds

all of the points fixed on a body. The visualization of an entire machine is the set of the convex hulls of all its bodies. A second option is to represent the bodies as *equivalent ellipsoids*. These are unique ellipsoids centered at the body's center of gravity, with the same principal moments of inertia and principal axes as the body.

More realistic renderings of bodies are possible, if the Virtual Reality Toolbox for Matlab is installed. Arbitrary virtual worlds can be designed with the Virtual Reality Modeling Language (VRML) and interfaced to the SimMechanics model.

3 An example problem

Some of the features of SimMechanics will now be shown on an example. The forward dynamics problem is solved with Matlab/ Simulink without the Physical Modeling environment as well as with SimMechanics. This allows a comparison of the results and the effort taken to get the desired solution. It is also shown briefly, how to compute the inverse dynamics and how to trim and linearize the model with SimMechanics. Finally, the problem of controlling an inverted pendulum is addressed as an example for the integrated design of mechanical and control systems.

3.1 The Mechanism

Figure 1 shows the mechanical system of interest. It consists of a uniform cubical rod B attached to a uniform cubical cart A which slides on a smooth horizontal plane in reference frame E. E can be oriented arbitrarily in the Newtonian reference frame N. A and B are connected by means of a frictionless revolute joint with the axis of rotation in $\mathbf{A}_3$ direction. A is attached to E with a linear spring/ damper system with the spring stiffness k_{SD} and the damping coefficient b_{SD}. The dimensions of the cart in $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$ direction are a_A, b_A, c_A, respectively, and the dimensions of the rod are similarly defined in reference frame B as a_B, b_B, c_B. The masses and central inertia dyadics of bodies A and B are assumed to be m^A, $\mathbf{I}^A$, m^B, and $\mathbf{I}^B$. The gravity force acts in $-\mathbf{N}_3$ direction. Two spatial forces, $\mathbf{F}^{A_*}$ and $\mathbf{F}^{P_1}$ act on the mechanism as shown. The system has one translational degree of freedom, q_1 and one rotational degree of freedom q_2.

The equations of motion for that mechanism can be derived by hand. They

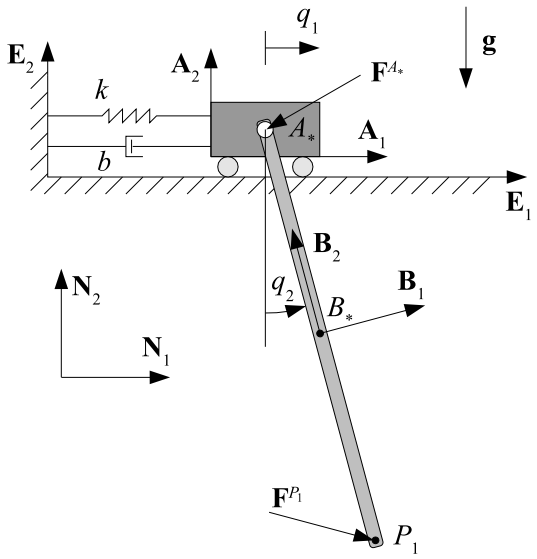

Figure 1: Pendulum on cart.

are given by

$$(m^A + m^B)\ddot{q}_1 + \frac{1}{2}m^B b_B \cos(q_2)\ddot{q}_2 + b_{SD}\,\dot{q}_1 - \frac{1}{2}m^B b_B \sin(q_2)\dot{q}_2^2 + k_{SD}\,q_1 - {}^E F_1^{A_*} - {}^E F_1^{P_1} = 0 \quad (1)$$

and

$$\frac{1}{2}m^B b_B \cos(q_2)\ddot{q}_1 + \frac{1}{4}(m^B b_b^2 + 4\,I_{33}^B)\ddot{q}_2 + \frac{1}{2}m^B g\,b_B \sin(q_2) - b_B\,{}^E F_1^{P_1} \cos(q_2) - b_B\,{}^E F_2^{P_1} \sin(q_2) = 0 \quad (2)$$

3.2 Forward Dynamics

As mentioned above, the forward dynamics have been computed with and without SimMechanics. The techniques to solve such problems in Matlab/ Simulink are not explained here explicitly, because the reader is assumed to be familiar with them.

For the forward dynamics problem, the external forces are set to $\mathbf{F}^{A_*} = \mathbf{F}^{P_1} = 0$. The initial conditions are given by $q_2 = 0.1\,\text{rad}$, and $q_1 = \dot{q}_1 = \dot{q}_2 = \ddot{q}_1 = \ddot{q}_2 = 0$. Figure 2 shows a m-file suitable to find the numerical solution. A Simulink block diagram for the same task is depicted in Figure 3. Both possibilities are implemented pretty easily once the equations (1) and (2) are

```
function [] = pocn_fdyn_ml(poc)
% POCN_FDYN_ML computes the forward dynamics of a pendulum o a cart.
% Input parameter is a structure 'poc', which is defined in the
% m-script 'poc_mdlvar.m'.
%
% written by: Michael Schlotter

% --------------------------------------------------------------------
% methods

tspan = [0:0.1:10]';              % timespan
y0 = [0 0.1 0 0]';                % initial conditions

options = odeset('Mass',@calc_m);
[t,y] = ode45(@odesystem,tspan,y0,options,poc);

output.time = t;
output.signals.dimensions = size(y,2);
output.signals.values = y;
assignin('base','yout_pocn_fkin_ml',output);
```

```
% ********************************************************************

function ydot = odesystem(t,y,poc)
% This subfunction presents the ode- system to the solver
% The mass matrix is state-dependent and is calculated in 'calc_m'.

ydot = [y(3); y(4);
  -poc.SD.b*y(3)+poc.B.m*0.5*poc.B.b*sin(y(2))*y(4)^2-poc.SD.k*y(1);
  -poc.B.m*poc.g*0.5*poc.B.b*sin(y(2))];

% ********************************************************************

function massmat = calc_m(t,y,poc)
% This subfunction calculates the state-dependent mass matrix.

massmat = eye(4);
massmat(3,3) = poc.A.m+poc.B.m;
massmat(3,4) = poc.B.m*0.5*poc.B.b*cos(y(2));
massmat(4,3) = poc.B.m*0.5*poc.B.b*cos(y(2));
massmat(4,4) = poc.B.m*(0.25*poc.B.b^2)+poc.B.I(3,3);
```

Figure 2: Matlab code for forward dynamics.

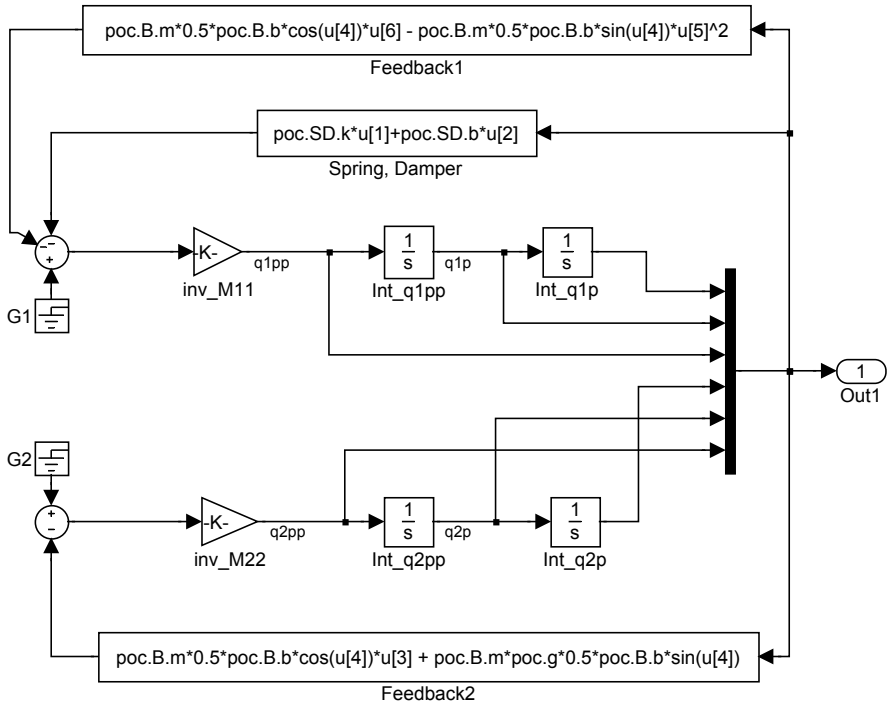

Figure 3: Simulink block diagram for forward dynamics.

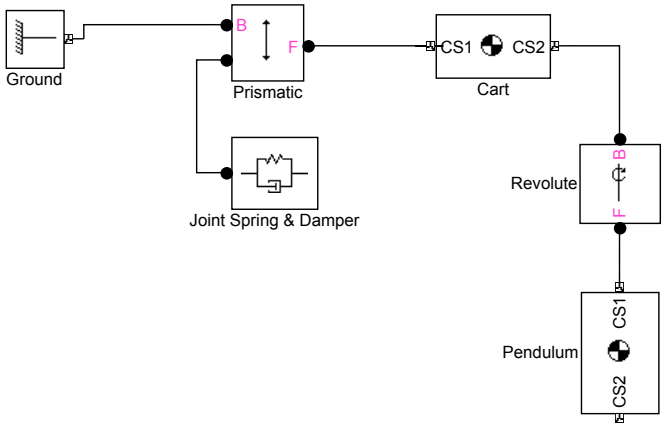

Figure 4: Simple SimMechanics block diagram for forward dynamics.

derived. This however has to be done by hand and is quite cumbersome and error prone.

The Physical Modeling environment SimMechanics makes the task much easier. The very simple block diagram of Figure 4 solves the problem without the need to derive any equations. Let us have a closer look at the diagram.

Obviously, every block corresponds to one mechanical component. The properties of the blocks can be entered by double-clicking on them. These are for example mass properties, dimensions and orientations for the bodies, the axis of rotation for the rotational joint and the spring/ damper coefficients for the Spring & Damper block. The initial conditions are given directly by specifying the initial position and orientations of the rigid bodies. With this model and the visualization facilities of SimMechanics it is possible to animate the motion of the pendulum and the cart.

As this model is so simple, it lacks some important features, though. It is not possible to specify nonzero initial conditions for velocities and accelerations, and no data is output to the workspace for further processing. This can be done by adding Sensor blocks and Joint Initial Condition blocks. With their help, a model which is functionally completely equivalent to the one in Figure 3 can be build. It is shown in Figure 5. The Joint Initial Condition blocks let the user define arbitrary initial conditions, and the Joint Sensor blocks measure the position, velocity, and acceleration of the two independent motion variables. If desired, the forces and torques transmitted by the joints can be sensed, too. Remarkable are the different types of connectors between the blocks. It is clearly visible that the input of the Sensor blocks are the special SimMechanics connection lines, while the output are normal Simulink signals which can be multiplexed and transfered to the Matlab workspace.

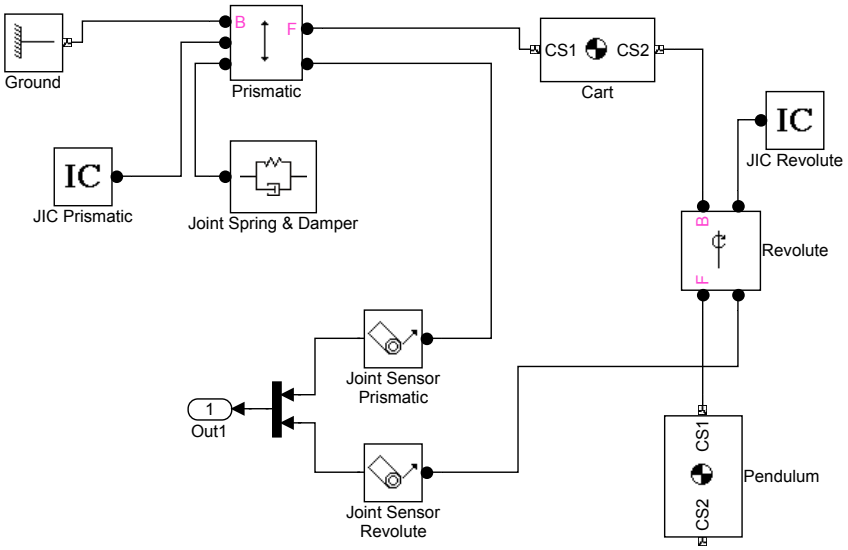

Figure 5: Complete SimMechanics block diagram for forward dynamics.

The results of all models are of course identical. Only the numerical errors differ slightly. The differences in the computation time are more obvious. As expected, the Matlab code is the fastest, followed by the Simulink model. The simulation with SimMechanics takes longer, because the mathematical model has to be derived by the software before the integration of the ODE system can begin. For such small models, however, computation time is mostly not an issue. More important is the time required to build the model, and in this discipline SimMechanics is unbeaten.

3.3 Visualization

SimMechanics is based on Simulink, and as that, all common options for exporting and plotting data are available. However, graphs like Figure 6 where the response of the system is shown by plotting q_1 and q_2 over the time t are rather abstract. If only the qualitative motion of a mechanism is of interest, the animation facilities of SimMechanics come into play. Figures 7 and 8 show the automatically generated animation windows if the options "equivalent ellipsoids" or "convex hulls" are chosen, respectively.

The representation with equivalent ellipses is only sensible, if all bodies have similar inertia dyadics. If one body is exceptionally big, it can be that it covers all others. An example is shown in Figure 9, where the inertia dyadic of the cart $\mathbf{I}^A$ is multiplied by 100. Unfortunately it is not yet possible to exclude such bodies from visualization.

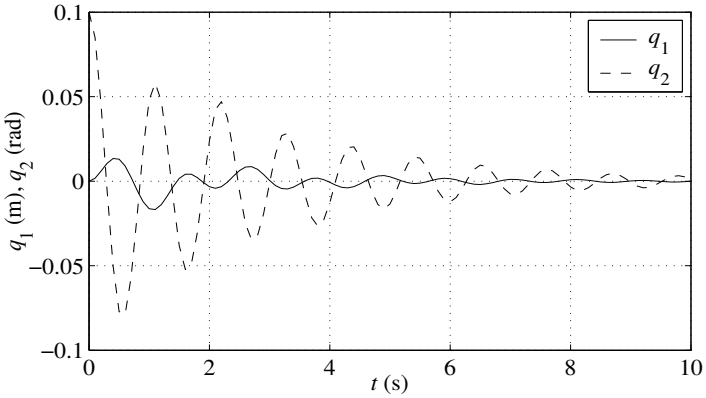

Figure 6: Response of the pendulum.

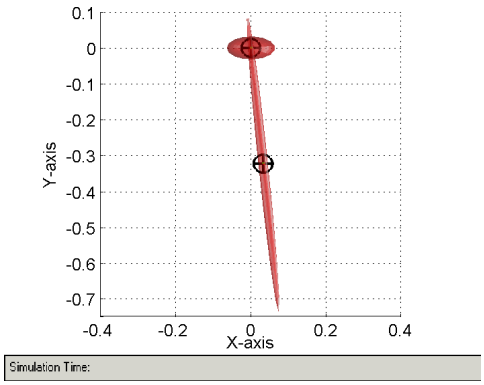

Figure 7: Animation display with option "equivalent ellipsoids".

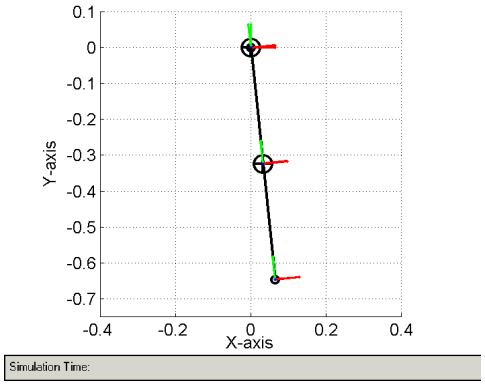

Figure 8: Animation display with option "convex hulls".

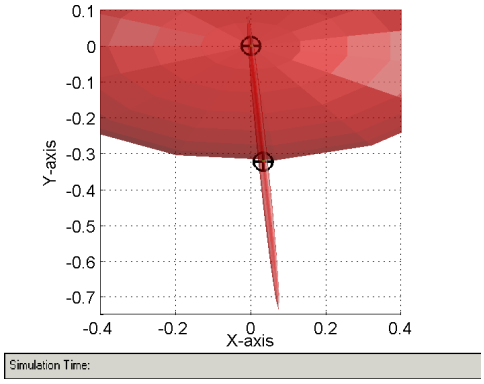

Figure 9: Equivalent ellipsoids view with a big body.

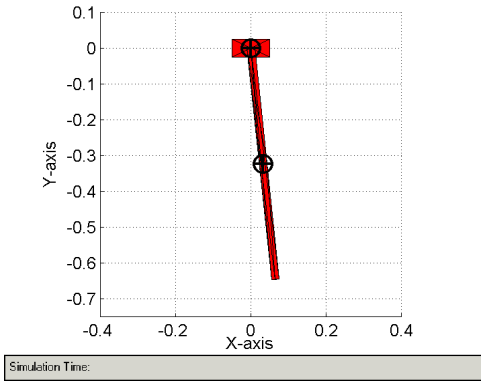

Figure 10: Enhanced convex hulls view.

The convex hulls option allows to improve the display by specifying additional fixed points on rigid bodies[1]. Figure 10 shows the result, if several points are added to the example model.

3.4 Inverse Dynamics

In Inverse Dynamics problems, the forces and torques required to let the system move as specified are the unknowns. Mostly, the results can be obtained by solving systems of nonlinear algebraic equations. In this discipline, SimMechanics underlies certain limitations, which becomes clear when looking at the following question. Considered $k_{SD} = b_{SD} = 0$, and $\mathbf{F}^{A_*} = F_A \cdot \mathbf{A}_1$, $\mathbf{F}^{P_1} = F_B \cdot \mathbf{B}_1$. What are $F_A(t)$ and $F_B(t)$, if the motion of the mechanism is given by $\dot{q}_1 = 1\,\mathrm{m/s} = const.$ and $\dot{q}_2 = 1\,\mathrm{rad/s} = const.$ with the initial conditions $q_1(t = 0) = q_2(t = 0) = 0$?

Although the solution can be obtained easily by hand with equations (1) and (2), there is no direct way to compute the forces directly with SimMechanics. To get a model suitable for inverse dynamics, the translational motion of the cart and the rotational motion of the pendulum have to be specified. This can be done with Joint Driver Actuator blocks. With this model, only the forces/ torques transmitted by the joints can be computed a a function of time. That is a force $\mathbf{F}_P$ which acts on A_* by the prismatic joint, and a torque $\mathbf{T}_R$ from A on B, transmitted by the revolute joint. The physically circumstance, that the revolute joint is frictionless (thus ${}^E T_3^{B/A} = 0$) and that only a force $\mathbf{F}^{P_1} = F_B \cdot \mathbf{B}_1$ drives the rotational degree of freedom can not be represented properly[2]. Mathematically it is not a problem, as the calculated forces and torques can be easily transformed to finally get the desired functions $F_A(t)$ and $F_B(t)$, as shown in Figure 11. Nevertheless certain limitations of the Physical Modeling approach become obvious.

3.5 Linearization, Trimming

The linearization and trimming functionalities for SimMechanics models are equivalent to traditional Simulink systems. The `linmod` command derives a

[1] Actually it is not possible to specify points. A workaround is to add coordinate systems fixed on the body, whose origins are considered as fixed points.

[2] It is not stated that it is not possible at all. By designing a closed loop system with additional (massless) bodies, joints, constraints, the problem can probably be solved. From the physical point of view this is even worse than the current solution.

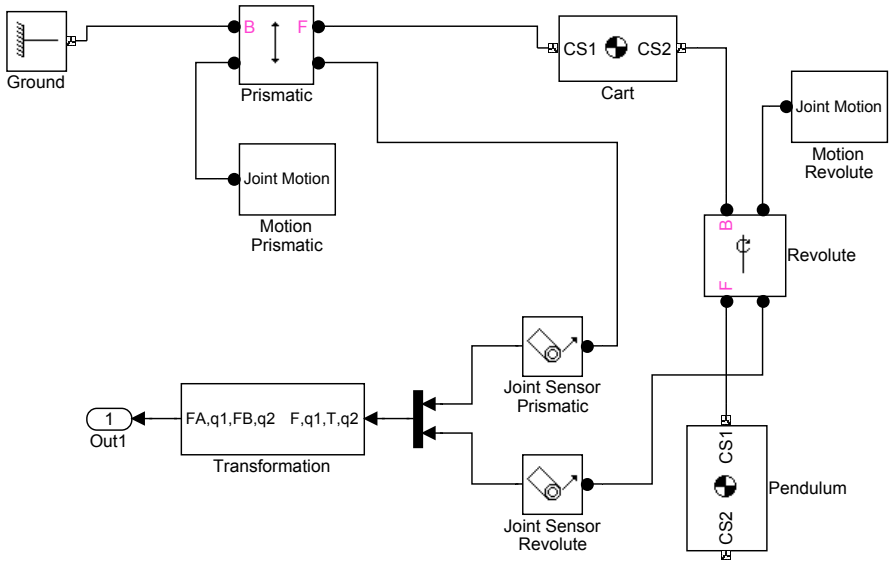

Figure 11: SimMechanics block diagram for inverse dynamics.

LTI state space model in the form

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{3}$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \tag{4}$$

where $\mathbf{x}$ is the model's state vector, $\mathbf{y}$ is its outputs, and $\mathbf{u}$ is its inputs.

The `trim` command finds solutions for the model that satisfy specified conditions on its inputs, outputs, and states, e.g. equilibrium positions where the system does not move, or steady state solutions where the derivatives of the system's states are zero.

In order to use both commands, the initial states of the system have to be specified. The new command `mech_stateVectorMgr` is of great help, as it can be used to displays the layout of the system's state vector, and the units in which the physical quantities have to be specified. For the forward kinematics model of Figure 5, for example, one gets for the state vector

```
>> v = mech_stateVectorMgr('pocn_fdyn_sm/Ground');
>> v.StateNames
ans =
    'pocn_fdyn_sm/Revolute:R1:Position'
    'pocn_fdyn_sm/Prismatic:P1:Position'
    'pocn_fdyn_sm/Revolute:R1:Velocity'
    'pocn_fdyn_sm/Prismatic:P1:Velocity'
```

which implies that $\mathbf{x} = (\dot{q}_2, \dot{q}_1, \ddot{q}_2, \ddot{q}_1)^T$.

All models designed for forward dynamics can be linearized. For trimming, to the contrary, one has to enter the Trimming mode. This causes SimMechanics to insert a subsystem into the model in order to deal with the position and motion constraints.

Everybody who is familiar with the Simulink `linmod` and `trim` commands will have no problems to use them with SimMechanics models. That is why they are not described here further, as this paper concentrates on the novelties introduced with SimMechanics.

3.6 Inverted pendulum

Starting from the forward dynamics model, it is straightforward to implement a classical control problem: the inverted pendulum on a cart. This study completes the example problem, because it shows how mechanical systems can be interfaced with control system models. As in the inverse dynamics problem, it is again assumed that $k_{SD} = b_{SD} = 0$, and $\mathbf{F}^{A_*} = F_A \cdot \mathbf{A}_1$. On the tip of the rod acts a perturbation force $\mathbf{F}^{P_1} = F_B \cdot \mathbf{B}_1$. The block diagram of Figure 12 reveals that the changes which have to be made on the forward dynamics model are minimal. Basically the Spring & Damper block is removed by a Joint Actuator block, and the initial position of the rod is of course changed so that it points upwards. The controllers are simple PD cascade controllers which implement the control law $F_A = K_p^P q_1 + K_d^P \dot{q}_1 + K_p^R q_2 + K_d^R \dot{q}_2$. The linearization facilities in connection with the Control Design Toolbox are a great help to find appropriate values for the controller constants. For $K_p^P = 25$, $K_d^P = 50$, $K_p^R = -500$, $K_d^R = -75$, where q_1 and $\dot{q}_1$ are measured in meters, and meters per second, and q_2 and $\dot{q}_2$ in radians, and radians per second, respectively, the following commands show that the system is stable for small perturbations, because the eigenvalues of the system matrix are all in the left half plane.

```
>> [A B C D] = linmod('poci_fdyn_pd');
>> eig(A)
ans =
  1.0e+002 *
  -2.0456
  -0.0831
  -0.0054 + 0.0053i
  -0.0054 - 0.0053i
```

This can be verified by plotting the time response of the system for an impulse force $\mathbf{F}^{P_1} = F_B \cdot \mathbf{B}_1$, as shown in Figure 13.

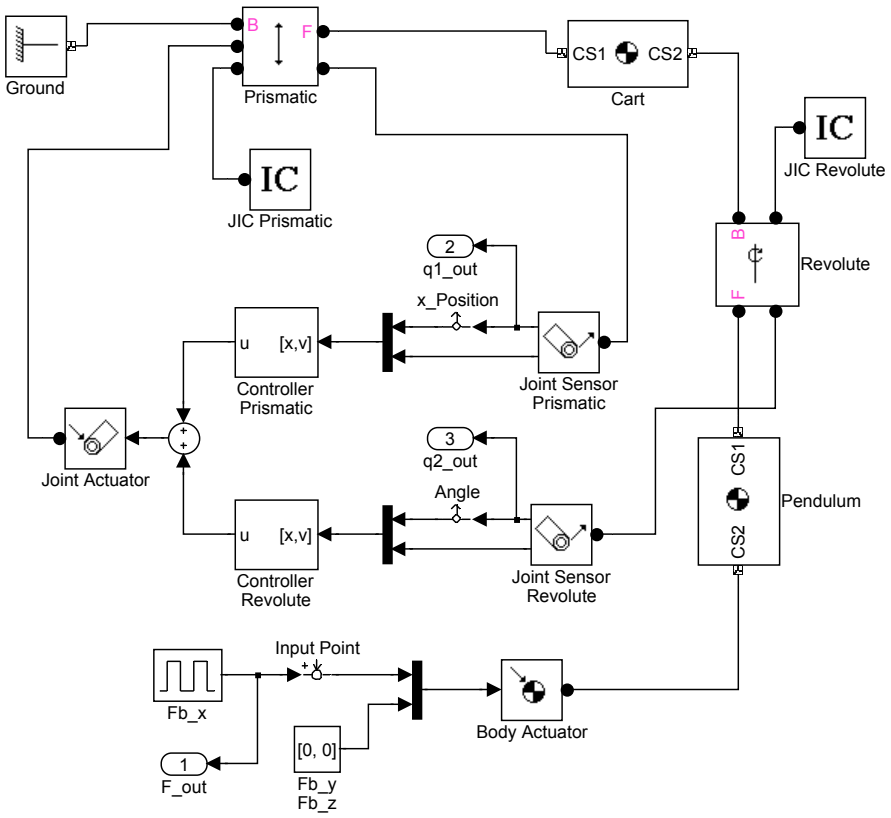

Figure 12: Interfacing SimMechanics systems with control system models.

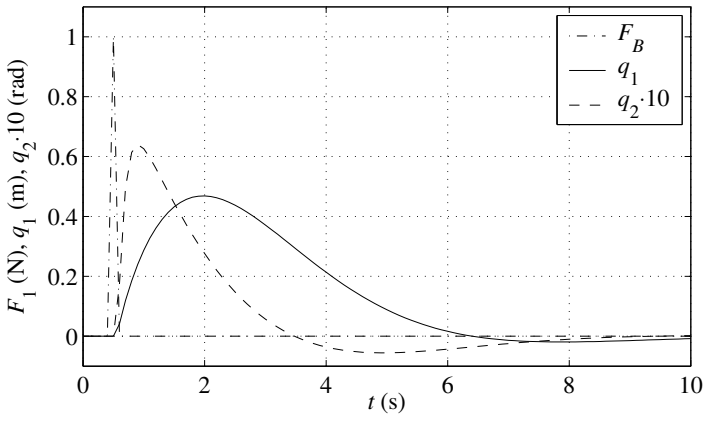

Figure 13: Response of the inverted pendulum to a impulse force.

4 Mathematical Aspects

In this section it is examined how SimMechanics works. Some interesting mathematical aspects are addressed, but the underlying algorithms are not described. The interested reader should refer to [Woo03] and the numerous references.

4.1 Equations of Motion for Multibody Systems

Although there are various derivations and formulations of the equations of motion for multibody systems, they can always be written as

$$\dot{\mathbf{q}} = \tilde{\mathbf{H}}(\mathbf{q})\mathbf{v} \tag{5}$$

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{v}} = \mathbf{f}(t, \mathbf{q}, \mathbf{v}) + \tilde{\mathbf{H}}^T(\mathbf{q})\mathbf{G}^T(\mathbf{q}, t)\boldsymbol{\lambda} \tag{6}$$

$$\mathbf{g}(\mathbf{q}, t) = 0 \tag{7}$$

Equation (5) expresses the kinematic relationship between the derivatives of the configuration variables $\mathbf{q}$ and the velocity variables $\mathbf{v}$. In the most simple cases $\tilde{\mathbf{H}}$ is the identity matrix. The second equation (6) is the motion equation with the positive definite mass matrix $\mathbf{M}$, the acceleration $\dot{\mathbf{v}}$, the contribution of the centrifugal, Coriolis, and external forces $\mathbf{f}$, and the contribution of reaction forces due to kinematical constraints which is expressed by the last term on the right side. Finally, equation (7) represents kinematical constraints between the configuration variables $\mathbf{q}$.

The main problem arising from equations (5)–(7) is that they form an index-3 differential algebraic equation[3] (DAE) because of the constraints in equation (7). Currently, Simulink is designed to simulate systems described by ODEs and a restricted class of index-1 DAEs, so the multibody dynamics problem is not solvable directly.

The structure of the equations of motion depends largely on the choice of coordinates. Many commercial software packages for multibody dynamics use the formulation in *absolute coordinates*. In this approach, each body is assigned 6 degrees of freedom first. Then, depending on the interaction of bodies due to joints, etc. suitable constraint equations are formed. This results in a large number of configuration variables and relatively simple constraint equations, but also in a sparse mass matrix $\mathbf{M}$. This sparseness and the uniformity in which the equations of motions are derived is exploited by the software. However, due to the large number of constraint equations, this strategy is not suitable for SimMechanics. It uses *relative coordinates*. In this approach,

[3] The index of a DAE is one plus the number of differentiations of the constraints that are needed in order to be able to eliminate the Lagrange multiplier $\boldsymbol{\lambda}$.

a body is is initially given zero degrees of freedom. They are "added" by connecting joints to the body. Therefore, far fewer configuration variables and constraint equations are required. Acyclic systems can even be simulated without forming any constraint equations. The drawback of this approach is the dense mass matrix $\mathbf{M}$, which now contains the constraints implicitly, and the more complex constraint equations (if any).

4.2 Simulation of Acyclic and Cyclic Systems

In the simulation problem, the motion of a mechanical system is calculated based on the knowledge of the torques and forces applied by the actuators, and the initial state of the machine. It can be divided into two subproblems:

1. *The forward dynamics problem* in which the joint accelerations are computed given the actuator torques and forces.

2. *The motion integration problem* in which the joint trajectories are computed based on the given acceleration and conditions.

Several computational techniques have been developed for the forward dynamics problem. Most of them are recursive methods. Two popular ones are the $O(n^3)$ Composite Rigid Body Method (CRBM) and the $O(n)$ Articulated Rigid Body Method (ABM) which are derived in [PAK00]. Both have their idiosyncrasies. The ABM uses the sparsity structure of large systems with many bodies (> 7) efficiently, but produces more computational overhead than the CRBM which can be preferable for smaller systems. The computational cost of the forward dynamics problem is only half the story, however. The characteristics of the resulting equations also have an influence on the performance of the adaptive time-step ODE solvers of Simulink ("formulation stiffness"). The ABM often produces better results, especially for ill-conditioned problems [APC96].

For unconstrained systems, the application of the CRBM and ABM methods, and the following integration of the ODEs is straightforward. A simulation program, however, must be able to handle arbitrary mechanisms. In SimMechanics, cyclic systems are reduced to open topology systems by cutting joints, which are (mathematically) replaced by a set of constraint equations. They ensure, that the system with the cut joints moves exactly as the original cyclic mechanism. The user can choose if he wants to select the cut-set by himself or if he leaves this task to SimMechanics. Of course, the structure of the resulting equations depends largely on this choice. In all cases, however, the equations of motion of constrained systems are index-3 DAEs, which have to be transformed into ODEs in order to be solvable with the Simulink ODE solver suite.

The approach taken by SimMechanics is to differentiate the constraint equation (7) twice and solve for the Lagrange multiplier $\boldsymbol{\lambda}$. Near singularities of the mechanism, i.e. near points where the number of independent constraint equations in decreased and the solution for $\boldsymbol{\lambda}$ is no longer unique, numerical difficulties arise. To deal with this problem, the user the choose between two solvers. One, based on Cholesky decomposition (the default), which is generally faster, and one based on QR decomposition, which is more robust in the presence of singularities [Sha94].

The main implication when turning DAEs into ODEs by differentiating the constraint equations is the numerical drift. Although the acceleration constraints will be fulfilled, the accumulation of roundoff and truncation errors may cause a violation of the position and velocity constraints. Common schemes which address this problem are *coordinate partitioning* [Dam02], *coordinate projection* [BT99], and *stabilization* [ACR94], [ACPR94].

In the coordinate partitioning approach, the Lagrange multiplier $\boldsymbol{\lambda}$ is eliminated from analysis, and the variables are split into a independent set which corresponds to the degrees of freedom of the system, and a dependent set, which correspond to the number of kinematic constraints. The minimal set of independent coordinates can be integrated with the ODE solvers, and the dependent coordinates can be found by solving the constraint equations. In practice, it is hard to select the independent coordinates, and the sparsity of the mass matrix is destroyed. These are the reasons, why SimMechanics does not offer this option. Coordinate Projection is used after each time step. The computed solution $\tilde{\mathbf{q}}_n$ of time-step t_n is projected on the invariant manifold $\mathbf{q}_n$, given by $g(\mathbf{q}_n, t_n) = 0$. This prevents the solution from drifting away. Finally, stabilization is based on adding stabilization parameters to the reduced ODE, which makes it more attractive to the manifold. A major difficulty is the selection of the parameters; for many stabilization methods there is no general solution for this problem. Nevertheless, several approaches methods are available which are a good compromise between numerical cost and accuracy. The SimMechanics user has the choice between the last two options. Coordinate projection is more exact, while the stabilization algorithms are faster and suitable for real-time applications.

Another aspects which is relevant for cyclic systems is the check for redundant constraints. It is performed once before simulation start. If redundant constraints occur, e.g. because of inconsistent motion drivers, SimMechanics can detect that by examining the numerical rank of the Jacobian $\mathbf{G}$ via QR decomposition (basically the same method as the one to detect and handle singularities). If the motion drivers are brought sequentially into analysis, it is even possible to detect the motion drivers which cause inconsistency. If the initial configuration is singular, however, redundant constraints may not be

detected before simulation start.

Also at simulation start, the initial conditions are checked for consistency. If a machine with a cyclic topology has to be simulated, the user can either build model with fully defined (possibly inconsistent) initial conditions, or he can use Disassembled Joint blocks which are assembled automatically by SimMechanics. In both ways, the software must be able to compute consistent initial conditions by solving systems of nonlinear equations. This is done with modified robust Gauss-Newton solvers and homotropy continuation solvers. The assembly tolerances can be specified by the user.

As already mentioned in Section 2.1, SimMechanics provides a Joint Stiction Actuator block. This block can cause joint locking and therefore discrete topology (or mode) changes. A special feature of mechanical systems is that multiple joints can lock at one instant of time, or that the locking of one joint causes another one to unlock. The detection of such events is done by mode iteration. In this approach, the topology change is introduced, and the system analyzed if the mode change causes any inconsistencies. If that is the case, an iteration is started to find a mode which is consistent with the applied torques and forces. After the right mode is found, cyclic systems are checked for constraint redundancies (without sequential addition of motion drivers to save computation time), and the integration resumes.

Since version 2, SimMechanics can be used for real-time Hardware in the loop (HIL) simulations. A major demand on such models is that they are deterministic, in the sense that the time to integrate one (fixed-size) time step must be predictable. Therefore, iterations which can take arbitrary time must be avoided. The consequence is that only the less accurate stabilization methods can be used to avoid numerical drift, and that the mode iteration is not carried out before the simulation resumes, but is spread over multiple time-steps.

4.3 Mathematics of Inverse Dynamics

Most of the aspects discussed in Section 4.2 also apply for inverse dynamics calculations. The difference is, that in this mode the equations of motion need not be integrated, but solved for the unknown reaction torques and forces, and the torques and forces generated by the constraint drivers. In general, this computation takes less time, as it is only an algebraic problem. Unfortunately, no detailed information about the inverse dynamics solvers used by SimMechanics could be found. It can be assumed, however, that recursive methods similar to CRBM and ABM are used.

4.4 Major Steps of the Dynamical Analysis

This section summarizes the steps carried out by SimMechanics during machine simulation. It is base on Chapter 5 of [Mat02].

1. *Model Validation*
   - Check of data entries in dialog boxes.
   - Validation of body, joint, constraint, driver geometry, and model topology.
2. *Machine Initialization*
   - Check of assembly tolerances.
   - Cutting of closed loops, formulation of constraint equations, and check for consistency and redundancy of constraints.
   - Examination of Joint Initial Condition blocks (if any), assembly of disassembled joints, and again check of assembly tolerances.
   - Mode iteration for sticky joints.
3. *Analysis*
   - Forward Dynamics/ Trimming modes:
     Application of external torques and forces. Formulation and integration of the equations of motion and solution for machine motion.
   - Inverse Dynamics/ Kinematics modes:
     Application of motion constraints, drivers, actuators. Formulation of equations of motion and solution for machine motion and actuator torques and forces.
   - After each time step in all analysis modes:
     Check of assembly, constraint, and solver tolerances, constraint and driver consistency and joint axis singularities.
   - If stiction is present:
     Stiction mode iteration, and check for constraint redundancy.

5 Conclusion

In this paper, the Matlab toolbox SimMechanics was introduced, it basic functionalities were shown on an example, and some mathematical aspect were addressed. The final judgment is very satisfactory. SimMechanics is an interesting and important add-on to the simulation environment Simulink. It allows

to easily include mechanical subsystems into Simulink models, without the need to derive the equations of motion. Especially non-experts will benefit from the visualization tools, as they facilitate the modeling and the interpretation of results.

SimMechanics is certainly not a replacement for specialized multibody dynamics software packages due to the limitations of the physical modeling approach and the restrictions of the Simulink environment. It is, however, well suited for many practical problems in education and industry, where Matlab became the standard language of technical computing.

References

[ACPR94] Uri M. Ascher, Hongsheng Chin, Linda R. Petzold, and Sebastian Reich. Stabilization of constrained mechanical systems with DAEs and invariant manifolds. Technical report, Department of Computer Science, University of British Columbia, Vancouver, Canada, October 1994. ftp://ftp.cs.ubc.ca/ftp/local/ascher/papers/acpr.ps.gz.

[ACR94] Uri M. Ascher, Hongsheng Chin, and Sebastian Reich. Stabilization of DAEs and invariant manifolds. Technical report, Department of Computer Science, University of British Columbia, Vancouver, Canada, November 1994. ftp://ftp.cs.ubc.ca/ftp/local/ascher/papers/acr.ps.gz.

[APC96] Uri M. Ascher, Dinesh K. Pai, and Benoit P. Cloutier. Forward dynamics, Elimination Methods, and Formulation Stiffness in Robot Simulation. Technical report, Department of Computer Science, University of British Columbia, Vancouver, Canada, May 1996. ftp://ftp.cs.ubc.ca/ftp/local/ascher/papers/apc.ps.gz.

[BT99] Claus Bendtsen and Per Grove Thomsen. Numerical Solution of Differential Algebraic Equations. Technical Report IMM-REP-1999-8, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 1999. www.imm.dtu.dk/documents/ftp/tr99/tr08_99.pdf.

[Dam02] Michael Damsgaard. *An Introduction to Dynamics of Rigid-Body Systems — Draft Version*, chapter 4. Institute of Mechanical Engineering, Aalborg University, October 2002. http://www.ime.auc.dk/people/employees/md/LectureNotes.htm.

[Mat02] The MathWorks, Inc. *SimMechanics User's Guide*, November 2002. http://www.mathworks.com.

[PAK00] Dinesh K. Pai, Uri M. Ascher, and Paul G. Kry. Forward Dynamics Algorithms for Multibody Chains and Contact. In *Proceedings of the IEEE Conference on Robotics and Automation*, 2000. http://www.cs.ubc.ca/~pai/papers/PaAsKr00.pdf.

[Sch02] Michael Schlotter. Dynamical Modeling and Analysis of the Canterbury Satellite Tracker. Project report, University of Canterbury, November 2002.

[Sha94] Ahmed A. Shabana. *Computational Dynamics*. Wiley-Interscience, 1994.

[Woo03] Giles D. Wood. Simulating Mechanical Systems in Simulink with SimMechanics. Technical report, The MathWorks, Inc, 2003. http://www.mathworks.com.

Appendix

A SimMechanics Model of the Canterbury Satellite Tracker

In order to check how good SimMechanics performs with models of "real" mechanisms, a simplified model of the Canterbury Satellite Tracker has been build. Details about this robot, shown in Figure 14, can be found in [Sch02].

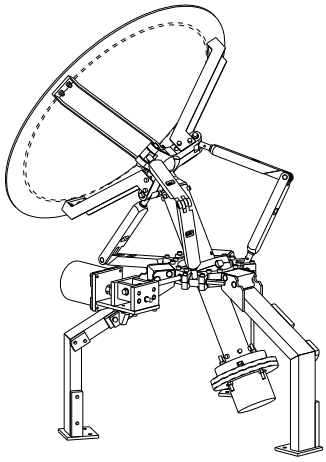

Figure 14: The Canterbury Satellite Tracker.

It was impressive to realize, how fast and easy the tracker mechanism could be implemented. Building the original Autolev model required good knowledge about the virtual force method (also known as *Kane's method*), and the formulation of configuration and motion constraints was quite difficult. Furthermore the generated calculation routines have often been unstable, and they still have some numerical difficulties at certain configurations of the mechanism. In SimMechanics the user does not need to bother about special dynamical methods or the formulation of constraint equations. The software does the job. Even the unsymmetric case[4] for which no analytic solution was found yet, was solved by SimMechanics without problems. Another nice feature is that reaction torques and forces in the joints can be sensed easily, which allows to use more advanced friction models, or to optimize the design of the joints.

As mentioned in section 3, the physical modeling approach of SimMechanics has some limitations, which require special attention. In this model, the op-

[4] *Unsymmetric* means that the six clevis arms which connect the lower circle with the upper dish holder may have different lengths.

tions to actuate disassembled and spherical joints were missed. Due to this limitation, modeling friction in such joints becomes tricky. However, this toolbox is very new and some features will certainly be added in future releases. Even now, SimMechanics proved to be a very suitable tool for multibody dynamics which allows the design of complex models in a fraction of time compared to other programs.

Excerpts of the block diagram for the tracker are shown in Figures 15, 16, and 17.

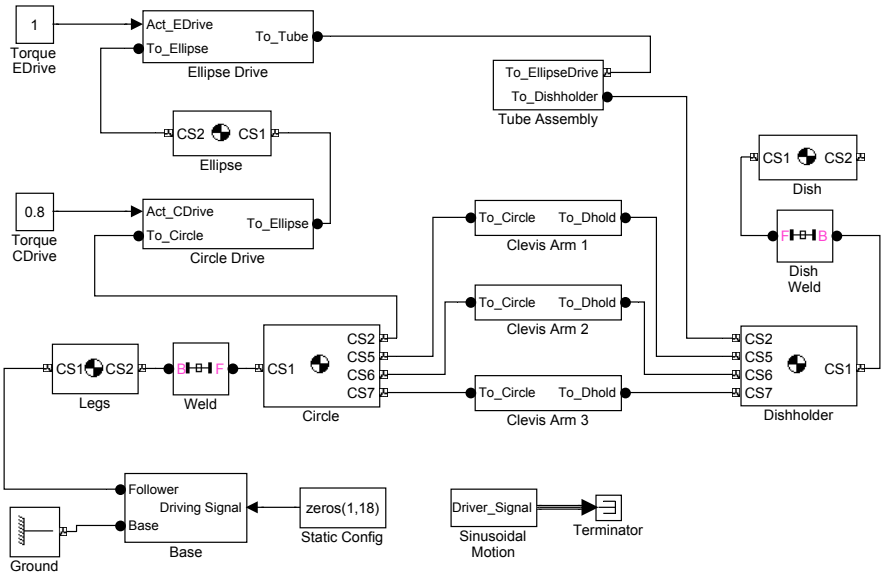

Figure 15: Top level of the block diagram.

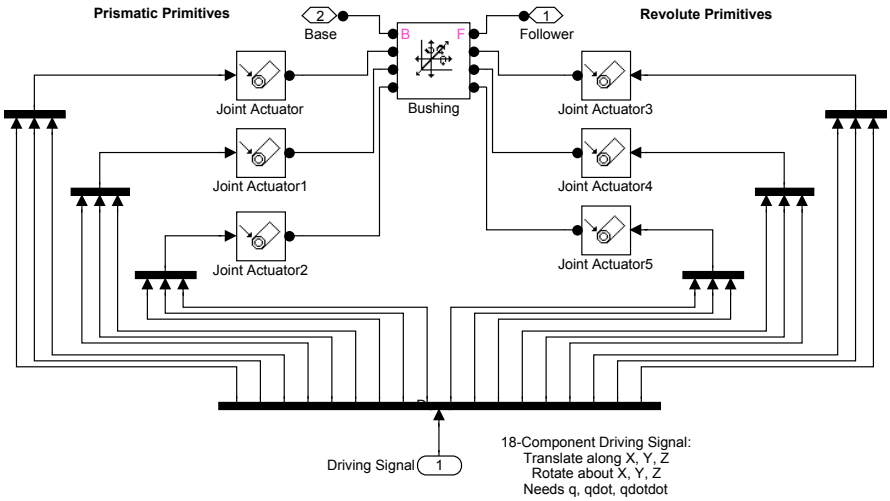

Figure 16: Implementation of the moving base. Sometimes one has to be a bit flexible when selecting joint blocks. The base is certainly not a *Bushing*, but this block contains all needed functionality.

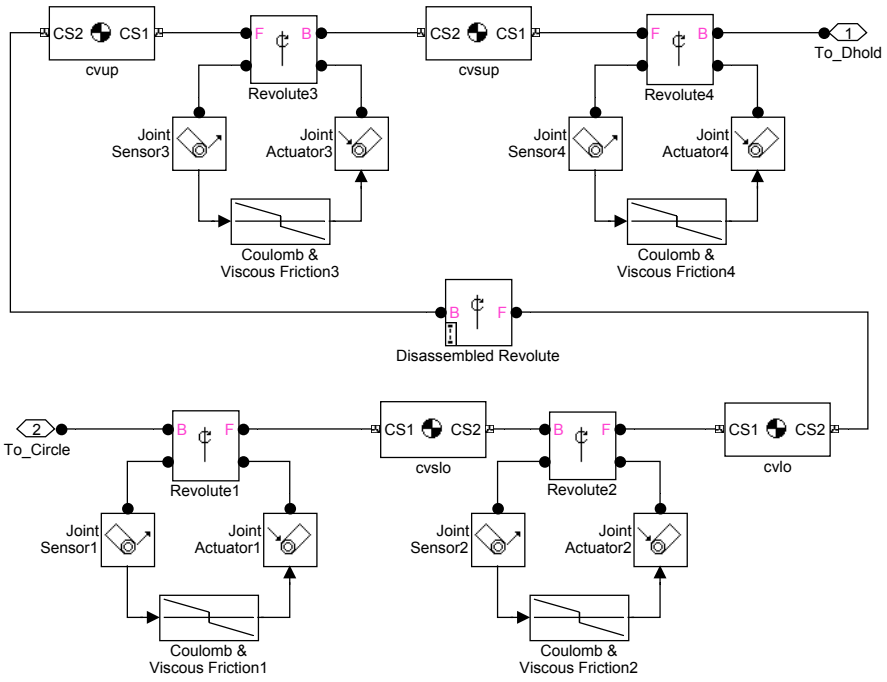

Figure 17: One of the three clevis arms. Note that the disassembled joint in the middle cannot be sensed or actuated.